第７章　mpi_c デーモン

7.1 はじめに

本章では、mpi_c デーモンに関する説明を行う。分散型の並列処理システムを構築するために、マスターから次々とスレーブとなる動的ソルバーを立ち上げる必要がある。この要求に答えるため、このデーモンを PC 上に常駐させ、他の PC からの接続を待ち受ける状態にさせておく。他の PC と接続した後は、接続した PC 上の動的ソルバーから情報を受信し、その情報を元に、この PC 上で動作するスレーブ用動的ソルバーを起動させる。この仕事が mpi_c デーモンに与えられた仕事の全てである。

7.2 デーモンの機能と使用法

分散並列型動的解析システムを構築するためには、まず、mpi_c デーモンを立ち上げておく必要がある。このデーモンは、手動で起動させても良いし、スタートアップメニューに登録して、Windows 起動時に自動的に立ち上がるようにしておいても良い。このソフトは、非常に小さいプログラムなので、常駐していても他のシステムに影響を与えることはない。

デーモンが起動していると、画面下のタスクバー右のパレットの中にアイコンが表示される。分散並列型動的解析システムを構築するためには、常にこの状態にしておかなければならない。このデーモンを終了するためには、次の手続きによる。まず、このアイコン上でマウスを右クリックし、プルダウンメニューを出現させる。このメニューの中の「ソフトの終了」を選択する。これで、デーモンを終了させることができる。

7.3 mpi_c_deamon

デーモンシステムは、以下のファイルで構成されている。

mpi_c_deamon.h	mpi_c_deamon.dsp
MainFrm.h	MainFrm.cpp
sock_func.h	sock_func.cpp

これらの機能は、以下のようである。

```
==========================================================================
        MICROSOFT FOUNDATION CLASS ライブラリ : mpi_c_deamon
==========================================================================
mpi_c_deamon.h  mpi_c_deamon.cpp
    このファイルmpi_c_deamon.hはアプリケーションの中心となるインクルードフ
```

ァイルである。このファイルは他のプロジェクトの固有のインクルードファイル（Resource.h　も含みます） をインクルードし、また CMpi_c_deamonApp アプリケーションクラスを宣言する。また、このファイルmpi_c_deamon.cppは CMpi_c_deamonApp アプリケーションクラスを含むアプリケーションの中心となるソースファイルである。

MainFrm.h　MainFrm.cpp
これらのファイルはフレーム クラス CMainFrame を含む。フレーム クラス CMainFrame は CFrameWnd クラスから派生し、SDI のフレームを制御する。

StdAfx.h　StdAfx.cpp
これらのファイルはプリコンパイル済ヘッダー ファイ(PCH)mpi_c_deamon.pch やプリコンパイル済型ファイル StdAfx.obj を構築するために使われるファイルである。

sock_func.h sock_func.cpp
これらのファイルはWinsock関数を集めたもので､このデーモン用に多少変更している。

最初に、mpi_c_deamon クラスについて説明する。以下には、ヘッダーファイルの mpi_c_deamon.h と resource.h、次に、mpi_c_deamon.cpp が示されている。ここでは、デーモンの機能についてのみ説明する。

mpi_c_deamon.cpp 内のコメント１のコードは､メインフレームのオブジェクトを作成し、2 では、ウインドウを表示する。ただし、引数として SW_HIDE を渡しており、実際にはウインドウは表示されない。3 では、ウインドウをアップデートする。このウインドウ表示で多くの処理が行われる。その操作については次節で説明する。

```
//    ――――――――――――――――――――――――――――――――――――――――
//      ●    mpi_c_deamon.h : アプリケーションのメイン ヘッダー ファイル
//    ――――――――――――――――――――――――――――――――――――――――
#if !defined(AFX_MPI_C_DEAMON_H__F1D658F9_DA0C_4A71_83CB_1BC2DE43467E__INCLUDED_)
#define AFX_MPI_C_DEAMON_H__F1D658F9_DA0C_4A71_83CB_1BC2DE43467E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"       // メイン シンボル

/////////////////////////////////////////////////////////////////////////////
// CMpi_c_deamonApp:
// このクラスの動作の定義に関しては mpi_c_deamon.cpp ファイルを参照してください。
//

class CMpi_c_deamonApp : public CWinApp
{
```

```
public:
	CMpi_c_deamonApp();

// オーバーライド
// ClassWizard は仮想関数のオーバーライドを生成します。
//{{AFX_VIRTUAL(CMpi_c_deamonApp)
public:
virtual BOOL InitInstance();
//}}AFX_VIRTUAL

// インプリメンテーション
//{{AFX_MSG(CMpi_c_deamonApp)
afx_msg void OnAppAbout();
// メモ - ClassWizard はこの位置にメンバ関数を追加または削除します。
//        この位置に生成されるコードを編集しないでください。
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ は前行の直前に追加の宣言を挿入します。

#endif // !defined(AFX_MPI_C_DEAMON_H__F1D658F9_DA0C_4A71_83CB_1BC2DE43467E__INCLUDED_)

//    ------------------------------------------------------------
//      ●   resource.h: 新規リソース ID を定義する標準ヘッダー ファイル
//    ------------------------------------------------------------
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by mpi_c_deamon.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_MPI_C_TYPE                  129
#define ID_MENUITEM32771                32771
#define ID_MENU_EXIT                    32772
//
// Next default values for new objects
//
#ifdef  APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                1
#define _APS_NEXT_RESOURCE_VALUE        130
#define _APS_NEXT_COMMAND_VALUE         32773
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif


//    ------------------------------------------------------------
//      ●   mpi_c_deamon.cpp : アプリケーション用クラスの機能定義
//    ------------------------------------------------------------
```

```
#include "stdafx.h"
#include "mpi_c_deamon.h"
#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMpi_c_deamonApp

BEGIN_MESSAGE_MAP(CMpi_c_deamonApp, CWinApp)
//{{AFX_MSG_MAP(CMpi_c_deamonApp)
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
// メモ - ClassWizard はこの位置にマッピング用のマクロを追加または削除します。
// この位置に生成されるコードを編集しないでください。
//}}AFX_MSG_MAP
// 標準のファイル基本ドキュメント コマンド
ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
// 標準の印刷セットアップ コマンド
ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

//    ―――――――――――――――――――――――――――――――――――――――
//      ●    CMpi_c_deamonApp クラスの構築
//    ―――――――――――――――――――――――――――――――――――――――
CMpi_c_deamonApp::CMpi_c_deamonApp()
{
}

/////////////////////////////////////////////////////////////////////////////
// 唯一の CMpi_c_deamonApp オブジェクト

CMpi_c_deamonApp theApp;

//    ―――――――――――――――――――――――――――――――――――――――
//      ●    InitInstance   CMpi_c_deamonApp クラスの初期化
//    ―――――――――――――――――――――――――――――――――――――――
BOOL CMpi_c_deamonApp::InitInstance()
{
       AfxEnableControlContainer();
#ifdef _AFXDLL
       Enable3dControls();         // 共有 DLL の中で MFC を使用する場合にはここを呼び出してください。
#else
       Enable3dControlsStatic();   // MFC と静的にリンクしている場合にはここを呼び出してください。
#endif
       // 設定が保存される下のレジストリ キーを変更します。
       // TODO: この文字列を、会社名または所属など適切なものに
       // 変更してください。
       SetRegistryKey(_T("Local AppWizard-Generated Applications"));
       LoadStdProfileSettings();     // 標準の INI ファイルのオプションをロードします (MRU を含む)
       m_pMainWnd = new CMainFrame;  // CMainFrameを直接操作します。                  // 1
```

```
        // メイン ウィンドウが初期化されたので、表示と更新を行います。
        m_pMainWnd->ShowWindow(SW_HIDE);                                    // 2
        m_pMainWnd->UpdateWindow();                                         // 3
        return TRUE;
}

//    ―――――――――――――――――――――――――――――――――――――――――――――――――
//      ●    CAboutDlg ダイアログ
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// ダイアログ データ
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA
        // ClassWizard 仮想関数のオーバーライドを生成します。
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV のサポート
        //}}AFX_VIRTUAL
// インプリメンテーション
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
//      ●    CDialog
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
//      ●    DoDataExchange
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
//      ●    OnAppAbout
//    ―――――――――――――――――――――――――――――――――――――――――――――――――
```

```
void CMpi_c_deamonApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

7.4 MainFrame

本節では、Mainfrm クラスについて説明する。このクラスは、次の 3 つの仕事を行う。

１）PC からの接続を受ける状態を作成する。
２）本来の仕事である他の PC からの接続を受け、同じ PC 内の動的ソルバーを起動させる。
３）メニューによって、このデーモンを終了させる。

Mainfrm クラスのヘッダーファイルを以下に示す。

```
//	―――――――――――――――――――――――――――――――――――――――――――
//	●　MainFrm.h : CMainFrame クラスの宣言およびインターフェイスの定義
//	―――――――――――――――――――――――――――――――――――――――――――
#if !defined(AFX_MAINFRM_H__C4310E92_9A12_4B13_B313_5F4BF1F54C00__INCLUDED_)
#define AFX_MAINFRM_H__C4310E92_9A12_4B13_B313_5F4BF1F54C00__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CFrameWnd
{
public:
	CMainFrame();

protected: // シリアライズ機能のみから作成します。
	DECLARE_DYNCREATE(CMainFrame)

// アトリビュート
public:
// オペレーション
public:
	void OnPopupMenu(CPoint point);		// ポップアップメニューを表示
	void OnAccept();			// 接続要求を受け入れ
// オーバーライド
	// ClassWizard は仮想関数のオーバーライドを生成します。
	//{{AFX_VIRTUAL(CMainFrame)
	public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);
```

```
        //}}AFX_VIRTUAL

// インプリメンテーション
public:
        virtual ~CMainFrame();

        SOCKET  m_hListen;                              // 待ち受け用ソケット        // 1
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // タスクトレイ用アイコンメンバ
        NOTIFYICONDATA      m_stNtfyIcon;

// 生成されたメッセージ マップ関数
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnDestroy();
        afx_msg void OnMenuExit();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ は前行の直前に追加の宣言を挿入します。

#endif // !defined(AFX_MAINFRM_H__C4310E92_9A12_4B13_B313_5F4BF1F54C00__INCLUDED_)
```

上記のヘッダーファイル内のコメント１に該当するコードで、待ち受け用のソケット m_hListen が作成される。

次に、CMainFrame クラスの本体を示す。このクラスの仕事のひとつを次に示す。前節で示したメインウインドウを表示する関数が実行すると、このクラスの中の OnCreate()関数が動作する。この関数の中で、2 で示す関数 sock_init()で、ソケットの初期設定が行われる。次に、3 の sock_listen()でポート PORT_DEAMON を用いて、TCP/IP 経由で他の PC からの接続を待つことになる。その関数の戻り値はソケットのハンドル m_hListen である。このハンドルは、他の PC よりメッセージを受け取る場合に必要となる。

```
//  ―――――――――――――――――――――――――――――――――――――――――
//    ●   MainFrm.cpp : CMainFrame クラスの動作の定義
//  ―――――――――――――――――――――――――――――――――――――――――
#include "stdafx.h"
#include "mpi_c_deamon.h"
#include "sock_func.h"
#include "MainFrm.h"
#include <string.h>
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define WM_USER_NOTIFYICON              (WM_USER + 100)
#define WM_SOCKET_LISTEN                (WM_USER + 200)
/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    メッセージマップ
//    ―――――――――――――――――――――――――――――――――――――――――
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
//{{AFX_MSG_MAP(CMainFrame)
ON_WM_CREATE()
ON_WM_DESTROY()
ON_COMMAND(ID_MENU_EXIT, OnMenuExit)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    CMainFrame クラスの構築
//    ―――――――――――――――――――――――――――――――――――――――――
CMainFrame::CMainFrame()
{
/////////////////////////////////
// ウィンドウクラスを登録する
WNDCLASS  stWndClass;
char  chClassName[] = _T("MPLDeamon");

stWndClass.style = CS_BYTEALIGNWINDOW;                  // スタイル
stWndClass.lpfnWndProc  = &AfxWndProc;                  // ウィンドウプロシージャのポインタ
stWndClass.cbClsExtra   = 0;
stWndClass.cbWndExtra   = 0;
stWndClass.hInstance    = AfxGetInstanceHandle();       // インスタンスハンドル
stWndClass.hIcon        = NULL;                         // アイコン
stWndClass.hCursor       = NULL;                        // カーソル
stWndClass.hbrBackground = (HBRUSH)COLOR_BACKGROUND;    // 背景色
stWndClass.lpszMenuName  = NULL;                        // メニューID
stWndClass.lpszClassName = chClassName;                 // ウィンドウクラス名
AfxRegisterClass(&stWndClass);                          // ウィンドウクラスを登録する
// 実際にウィンドウを作成する
Crect rect(0,0,100,100);
Create(chClassName, _T("mpi_c_deamon"), WS_OVERLAPPEDWINDOW  | WS_MINIMIZE, rect);// ウィンドウを構築

}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    CMainFrame クラスの消滅
//    ―――――――――――――――――――――――――――――――――――――――――
CMainFrame::~CMainFrame()
{
```

```
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    OnCreate
//    ―――――――――――――――――――――――――――――――――――――――――――――
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
return -1;
////////////////////////////////
//  アイコンをタスクトレイに表示。
m_stNtfyIcon.cbSize = sizeof(NOTIFYICONDATA);   // 構造体のサイズ
m_stNtfyIcon.uID    = 0;                        // アイコンの識別ナンバー
m_stNtfyIcon.hWnd   = m_hWnd;                   // メッセージを知らせるウィンドウのハンドル
m_stNtfyIcon.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP;    // 各種設定
m_stNtfyIcon.hIcon  = AfxGetApp()->LoadIcon(IDR_MAINFRAME);// アプリケーションのハンドル
m_stNtfyIcon.uCallbackMessage = WM_USER_NOTIFYICON;        // 送ってもらうメッセージ
lstrcpy(m_stNtfyIcon.szTip, _T("mpi_c Deamon"));           // チップの文字列
::Shell_NotifyIcon(NIM_ADD, &m_stNtfyIcon);                // タスクトレイに表示
// 待ち受けを開始
sock_init();                                                                    // 2
m_hListen = sock_listen(PORT_DEAMON);                                           // 3
int nErr = WSAAsyncSelect(m_hListen, GetSafeHwnd(), WM_SOCKET_LISTEN, FD_ACCEPT);
return 0;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    PreCreateWindow
//    ―――――――――――――――――――――――――――――――――――――――――――――
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
if( !CFrameWnd::PreCreateWindow(cs) )  return FALSE;
return TRUE;
}

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
CFrameWnd::Dump(dc);
}

#endif //_DEBUG

//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    WindowProc
//    ―――――――――――――――――――――――――――――――――――――――――――――
LRESULT CMainFrame::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
switch(message)
{
case WM_USER_NOTIFYICON:         //  アイコンからのメッセージ
```

```
            if(lParam == WM_RBUTTONDOWN)                                    // 4
             {
               CPoint pt;
               GetCursorPos(&pt); // マウスカーソルの位置を取得
               OnPopupMenu(pt);
             }
             return 0;
case WM_SOCKET_LISTEN:          // 接続要求メッセージ
             OnAccept();                                                    // 5
             break;
             }
return CFrameWnd::WindowProc(message, wParam, lParam);
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    OnDestroy
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CMainFrame::OnDestroy()
{
CFrameWnd::OnDestroy();
::Shell_NotifyIcon( NIM_DELETE, &m_stNtfyIcon );        //タスクトレイのアイコンを削除します。
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    OnPopupMenu
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CMainFrame::OnPopupMenu(CPoint point)
{
SetForegroundWindow();    // ウィンドウをフォアグラウンドに持ってくる。
SetFocus();               // これをしないと、メニューが消えなくなる。
// ポップアップメニューを出す
// 右ボタンポップアップメニューの登録
CMenu menu;
VERIFY(menu.LoadMenu(IDR_MAINFRAME));
CMenu* pPopUp = menu.GetSubMenu(0);
pPopUp->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x, point.y, (CWnd*)this);
PostMessage( WM_NULL );     //これをしないと、２度目のメニューがすぐ消えてしまう。
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    OnMenuExit
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CMainFrame::OnMenuExit()
{
DestroyWindow();
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    OnAccept マスターあるいは、スレーブからのデータの受信
//    ―――――――――――――――――――――――――――――――――――――――――――――
void CMainFrame::OnAccept()
{
SOCKET hSocket;                                                             // 6
HINSTANCE hRet;
int nRet;
int message;
char szParam1[PARAM_SIZE], szParam2[PARAM_SIZE];          // パラメータ
char szPass[_MAX_DIR];                                    // 起動アプリのパス
```

```
char seps[] = "";
// 接続を受入れ
if(sock_accept(m_hListen, hSocket) == false)  return;                          // 7
// メッセージを受信
nRet = sock_recv(hSocket, &message, sizeof(message));       // メッセージ       8
nRet = sock_recv(hSocket, &szParam1, sizeof(szParam1));     // パラメータ１     9
nRet = sock_recv(hSocket, &szParam2, sizeof(szParam1));     // パラメータ２     10
// 受信したメッセージを応答する
switch(message)
{
// 起動要求
case MPI_MSG_BOOT:                                                             // 11
      // 起動パスを設定
      GetCurrentDirectory(_MAX_DIR, szPass);                // カレントディレクトリを取得
      strcat(szPass, "¥¥");
      strcat(szPass, szParam1);                             // アプリ名を設定
      // 起動オプションを設定
      // スレーブを起動
      TRACE("szPass = %s %s¥n", szPass, szParam2);
      hRet = ShellExecute(NULL, "open", szPass, szParam2, NULL, SW_SHOW);
      TRACE("hRet = %u¥n", hRet);
      break;
default:                                                                        // 12
      break;
      sock_close(hSocket);
}
```

デーモンがメッセージを待ち受けているとき、2 種類のメッセージがこのソケットに到達する。到達すると、関数 WindowProc()が実行され、そのメッセージの種類によって、2 つの仕事が行われる。マウスからのメッセージによって、4 で示す処理が実行され、ポップアップメニューを表示させる関数 OnPopupMenu(pt)が動作する。他の PC からの接続要求を示すメッセージは、5 で示す関数 OnAccept()が実行される。ここでは、先に示した第 2 の仕事を行う。

第 3 の仕事は、関数 OnPopupMenu()で行われる。メニューオブジェクトを作成し、表示させる。メニューは２つあり、「バージョン情報」と「ソフトの終了」。このうちのひとつを選択すると、次のメッセージが発生する。このメッセージは、「ID_APP_ABOUT」と「ID_MENU_EXIT」であり、これらにより下記のメッセージマップ内の関数によって、該当する関数が実行される。

```
mpi_c_deamon.cpp 内のメッセージマップ
ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
MainFrm.cpp 内のメッセージマップ
ON_COMMAND(ID_MENU_EXIT, OnMenuExit)
```

最初のメニューでは、mpi_c_deamon.cpp 内の関数 OnAppAbout()が実行される。この関数では、このデーモンのバージョン情報が表示される。次のメニューでは、MainFrm.cpp 内の関数 OnMenuExit()が実行される。この関数では、DestroyWindow()が実行され、デーモンが終了する。

第 2 の仕事は、5 で示す OnAccept()が実行される。この関数では、他の PC からのメッセージを受け取り、スレーブ用動的ソルバーを起動させる。それでは、ここでこの関数を詳細に見ていこう。以下に、プログラムのコメント番号に合わせて処理内容を説明する。

6. 各種のオブジェクトや変数を作る。ソケット hSocke は、メッセージを受けるためのハンドルである。
7. 関数 sock_accept()を用いて、接続を受け付ける。失敗した場合は、この関数より戻る。
8. メッセージを関数 sock_recv()を用いて、受信する。
9. パラメータ 1 として、起動するアプリケーション名を受信する。
10. パラメータ 2 として、このスレーブのランクを受信する。
11. メッセージが MPI_MSG_BOOT であると、スレーブ用動的解析システムを起動する処理を実行する。最初に、このソフトがカレントにあるとして、そのカレントの絶対パスを取得する。この絶対パスとアプリケーション名を結合して、スレーブ用の動的解析システムの絶対パス名を設定する。次に、関数 ShellExecute()を用いて、スレーブ用の動的解析システムを起動させる。この時、そのスレーブのランクをパラメータとして受け渡す。
12. メッセージがその他であると、データ受信用ソケットを閉じる。

7.5 winsock 関数

本節では、ソケット関数が示されている。この関数は、ライブラリー用に作成したものを一部変更して利用している。したがって、このデーモンシステムでは使用していない関数が見られる。関数の内容は、前章で説明した関数とほとんど同じであるので、理解することは難しくない。良く読んで理解されたい。

```
//    ―――――――――――――――――――――――――――――――――――――
//      ●    winsock関数の拡張
//    ―――――――――――――――――――――――――――――――――――――
#ifndef SOCK_FUNC_H
#define SOCK_FUNC_H
```

```
        bool sock_init();
        void sock_close(SOCKET hSocket);
        SOCKET sock_connect(const char* szServer, int nPort);
        SOCKET sock_listen(int nPort);
        bool sock_accept(SOCKET hListen, SOCKET &retSocket);
        int sock_send(SOCKET hSocket, const void* lpBuf, int nSize);
        int sock_recv(SOCKET hSocket, void* lpBuf, int nSize);
        bool get_local_IP(char* szAddress, int nSize);
// 各種定義
// ポートの定義
#define PORT_DEAMON         1950        // デーモンポート
#define PORT_PARALLEL       2000        // 基本ポート
// メッセージ
#define  MPI_MSG_BOOT       1000
#define  PARAM_SIZE         50

// 送信最大サイズ
#define MAX_SIZE            8000        // 一回のsendで送信可能な最大サイズ(Byte)

#endif // SOCK_FUNC_H

//    ―――――――――――――――――――――――――――――――――――――――
//      ●   winsock拡張関数の実装
//    ―――――――――――――――――――――――――――――――――――――――
#include "stdafx.h"
#include "sock_func.h"
#include "MPI_C_Core.h"
#include "mpi_c.h"
#include <stdlib.h>

// システムの実体
CMPI_C_Core core;
// コミュニケータの実体
MPI_Comm system_comm;
// 一時受信ステータスの実体
MPI_Status system_stat;
//    ―――――――――――――――――――――――――――――――――――――――
//      ●   MPI_Init
//    ―――――――――――――――――――――――――――――――――――――――
bool MPI_Init(int argc, char* argv[])
{
        int nID;
        if(argc == 2) return false;
        // 引数を取得 ※  argv[0]は実行ファイル名なので無視
        nID = atoi(argv[1]); // IDを取得
        return core.Init(nID, argv[2]);
}
//    ―――――――――――――――――――――――――――――――――――――――
//      ●   MPI_Init
//    ―――――――――――――――――――――――――――――――――――――――
bool _MPI_Init(int nID, const char* szParam)
{
        return core.Init(nID, szParam);
}
```

```
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    MPI_Send
//    ―――――――――――――――――――――――――――――――――――――――――――――――
int MPI_Send(const void *pBuf, int nCount, MPI_Datatype type, int nDest, int nTag, MPI_Comm comm)
{
       return core.Send(nDest, pBuf, get_size(nCount, type));
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    _MPI_Send
//    ―――――――――――――――――――――――――――――――――――――――――――――――
int _MPI_Send(const void *pBuf, int nSize, int nDest)
{
       return core.Send(nDest, pBuf, nSize);
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    MPI_Recv
//    ―――――――――――――――――――――――――――――――――――――――――――――――
int MPI_Recv(void *pBuf, int nCount, MPI_Datatype type, int nSource, int nTag,
                          MPI_Comm &comm, MPI_Status &rStatus)
{
       return core.Recv(nSource, pBuf, get_size(nCount, type));
}

//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    _MPI_Recv
//    ―――――――――――――――――――――――――――――――――――――――――――――――
int _MPI_Recv(void *pBuf, int nSize, int nSource)
{
       return core.Recv(nSource, pBuf, nSize);
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    MPI_Finalize
//    ―――――――――――――――――――――――――――――――――――――――――――――――
void MPI_Finalize()
{
       core.ShutDownAll();
}
//    ―――――――――――――――――――――――――――――――――――――――――――――――
//      ●    get_size
//    ―――――――――――――――――――――――――――――――――――――――――――――――
int get_size(int nCount, MPI_Datatype type)
{
       switch(type)
       {
       case MPI_INT:
              return sizeof(int) * nCount;
       case MPI_LONG:
              return sizeof(long) * nCount;
       case MPI_SHORT:
              return sizeof(short) * nCount;
       case MPI_UNSIGNED_SHORT:
              return sizeof(unsigned short) * nCount;
       case MPI_UNSIGNED:
              return sizeof(unsigned) * nCount;
```

```
        case MPI_UNSIGNED_LONG:
                return sizeof(unsigned long) * nCount;
        case MPI_FLOAT:
                return sizeof(float) * nCount;
        case MPI_DOUBLE:
                return sizeof(double) * nCount;
        case MPI_REAL:
                return sizeof(float) * nCount;
        case MPI_DOUBLE_PRECISION:
                return sizeof(double) * nCount;
        case MPI_LONG_DOUBLE:
                return sizeof(long double) * nCount;
        case MPI_COMPLEX:
                return sizeof(COMPLEX) * nCount;
        case MPI_BYTE:
                return sizeof(unsigned char) * nCount;
        default:
                return -1;
        }
}

//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    winsock拡張関数の実装
//    ―――――――――――――――――――――――――――――――――――――――――
#include "stdafx.h"
#include "sock_func.h"
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    sock_init
//    ―――――――――――――――――――――――――――――――――――――――――
bool sock_init()
{
        int nRet;
        WORD wVersionRequested = MAKEWORD(1,1);          // WinSockバージョン1.1
        WSADATA wsaData;                                 // WinSock情報
        // WinSockの初期化
        nRet = WSAStartup(wVersionRequested, &wsaData);
        if(wsaData.wVersion != wVersionRequested)
        {
                return false;
        }
        return true;
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    sock_close
//    ―――――――――――――――――――――――――――――――――――――――――
void sock_close(SOCKET hSocket)
{
        int nRet;
        char szBuf[1000];
        // 接続を切断する
        shutdown(hSocket, 1);              // 送信停止
        do                                 // 残っているデータを全て受信
        {
                nRet = recv(hSocket, szBuf, sizeof(szBuf), 0);
```

```
        }while(nRet > 0);
        shutdown(hSocket, 2);              // 受信停止
        closesocket(hSocket);              // 切断
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    sock_connect
//    ―――――――――――――――――――――――――――――――――――――――――――――
SOCKET sock_connect(const char* szServer, int nPort)
{
        SOCKET hSocket;
        LPHOSTENT lpHostEntry;             // サーバアドレス
        int nRet;

        // TCP/IPソケットを作成
        hSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if(hSocket == INVALID_SOCKET)
        {
                return INVALID_SOCKET;
        }

        // 相手を検索
        lpHostEntry = gethostbyname(szServer);
        if(lpHostEntry == NULL)
        {
                return INVALID_SOCKET;
        }

        // アドレス構造体を埋める
        SOCKADDR_IN saServer;
        saServer.sin_family = AF_INET;
        saServer.sin_addr = *((LPIN_ADDR)*lpHostEntry->h_addr_list);
        saServer.sin_port = htons(nPort);

        // 相手と接続
        nRet = connect(hSocket, (LPSOCKADDR)&saServer, sizeof(SOCKADDR));
        if(nRet == SOCKET_ERROR)
        {
                closesocket(hSocket);
                return INVALID_SOCKET;
        }
        return hSocket;
}
//    ―――――――――――――――――――――――――――――――――――――――――――――
//      ●    sock_listen
//    ―――――――――――――――――――――――――――――――――――――――――――――
SOCKET sock_listen(int nPort)
{
        int nRet;
        SOCKET hListen;
        hListen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if(hListen == INVALID_SOCKET)
        {
                return INVALID_SOCKET;
        }
```

```
        // アドレス構造体を埋める
        SOCKADDR_IN saServer;                           // サーバアドレス
        saServer.sin_family = AF_INET;
        saServer.sin_addr.s_addr = INADDR_ANY;
        saServer.sin_port = htons(nPort);

        // ソケットをアドレスに関連付ける
        nRet = bind(hListen, (LPSOCKADDR)&saServer, sizeof(SOCKADDR_IN));
        if(nRet == SOCKET_ERROR)
        {
                closesocket(hListen);
                return INVALID_SOCKET;
        }
        // クライアントからの受信を待つ
        listen(hListen, SOMAXCONN);         // WinSockにこのソケットで待ち受けすることを告げる
        if(nRet == SOCKET_ERROR)
        {
                closesocket(hListen);
                return INVALID_SOCKET;
        }
        return hListen;
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    sock_accept
//    ―――――――――――――――――――――――――――――――――――――――――
bool sock_accept(SOCKET hListen, SOCKET &retSocket)
{
        // 待ち受け用ソケットに接続があるまで待つ
        retSocket = accept(hListen, NULL, NULL);
        if(retSocket == INVALID_SOCKET)
        {
                closesocket(hListen);
                return false;
        }
        return true;
}
//    ―――――――――――――――――――――――――――――――――――――――――
//      ●    sock_send
//    ―――――――――――――――――――――――――――――――――――――――――
int sock_send(SOCKET hSocket, const void* lpBuf, int nSize)
{
        int nSizeSend;                      // 転送バイトサイズ
        int nErr;                           // エラー値
        PBYTE pBuf = (PBYTE)lpBuf;          // データのアドレス
        int nWritten;                       // 送信バイト数(戻り値)
        int nLeft = nSize;                  // 残りバイト数

        while(nLeft > 0)
        {
                // 転送サイズを設定
                if(nLeft > MAX_SIZE) nSizeSend = MAX_SIZE;
                else nSizeSend = nLeft;
                // 何かを送信するか、エラーが出るまで送信を続ける
                do
```

```
		{
			nWritten = send(hSocket, (LPSTR)pBuf, nSizeSend, 0);
			if(nWritten == SOCKET_ERROR)
			{
				nErr = WSAGetLastError();
				if(nErr != WSAEWOULDBLOCK)
				{
					return nWritten;
				}
			}
		} while(nWritten == SOCKET_ERROR);
		nLeft -= nWritten;			// 残りバイト数を計算
		pBuf += nWritten;			// データの送信位置を移動
	}
	return nSize - nLeft;
}
//	―――――――――――――――――――――――――――――――――――――――――
//	●　sock_recv
//	―――――――――――――――――――――――――――――――――――――――――
int sock_recv(SOCKET hSocket, void* lpBuf, int nSize)
{
	int nRead;				// 受信バイト数(戻り値)
	int nErr;				// エラー値
	PBYTE pBuf = (PBYTE)lpBuf;		// 保存先アドレス
	int nLeft = nSize;			// 残りバイト数

	// バッファの初期化
	memset(pBuf, 0, nSize);
	while(nLeft > 0)
	{
		// 何かを受信するか、エラーが出るまで受信を続ける
		do
		{
			nRead = recv(hSocket, (LPSTR)pBuf, nLeft, 0);
			if(nRead == SOCKET_ERROR)
			{
				nErr = WSAGetLastError();
				if(nErr != WSAEWOULDBLOCK)
				{
					return nRead;
				}
			}
		} while(nRead == SOCKET_ERROR);
		nLeft -= nRead;			// 残りバイト数を計算
		pBuf += nRead;			// データの受信位置を移動
	}
	return nSize - nLeft;
}
//	―――――――――――――――――――――――――――――――――――――――――
//	●　get_local_IP
//	―――――――――――――――――――――――――――――――――――――――――
bool get_local_IP(char* szAddress, int nSize)
{
	TCHAR szLocal[MAX_COMPUTERNAME_LENGTH + 1 ];
```

```
        DWORD dwRet = MAX_COMPUTERNAME_LENGTH + 1;
        LPHOSTENT lpHostEntry;              // hostent構造体のポインタ
        struct in_addr *pInAddr;            // インターネットアドレスを指すポインタ

        if(nSize < MAX_COMPUTERNAME_LENGTH + 1)
        {
              return false;
        }
        // ホストエントリを取得
        if(!GetComputerName(szLocal, &dwRet))
        {
              return false;
        }
        lpHostEntry = gethostbyname(szLocal);
        if(lpHostEntry == NULL)
        {
              return false;
        }
        // ＩＰアドレスを取得
        pInAddr = ((LPIN_ADDR)lpHostEntry->h_addr_list[0]);
        strcpy(szAddress, inet_ntoa(*pInAddr));
        return true;
  }
```